Fast Guid for Experiments

Chen Lequn

June 15, 2020

Contents

1 Establish PC-robot/sensor communication

1.1 MicroEpsilon

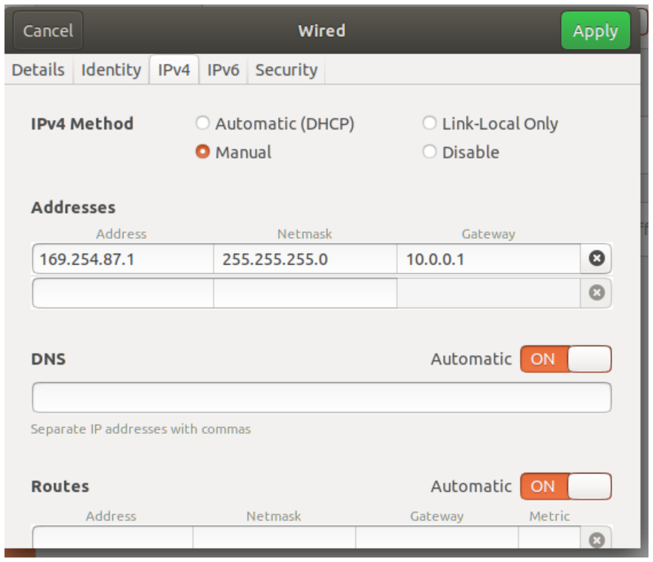
1.1.1 Ethernet settings:

- IP address: **169.254.87.67**
- serial number: 218120116
- type: 2950-50

1.1.2 configure ethernet setting on Ubuntu Linux (18.04)

- Configure Wired Setting As shown, specify address, gateware and netmask, so that PC can find the device
- execute command:

```
ping 169.254.87.67
```



IPv4 Method: Automatic (DHCP) [✓ Manual] Link-Local Only Disable

Addresses

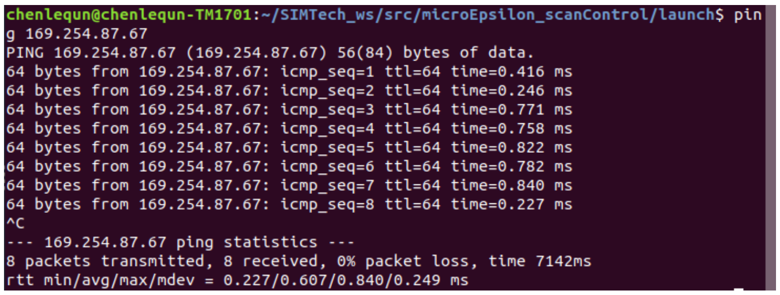
Address	Netmask	Gateway
169.254.87.1	255.255.255.0	10.0.0.1

DNS Automatic ON

Separate IP addresses with commas

Routes Automatic ON

Address Netmask Gateway Metric



```
chenlequn@chenlequn-TM1701:~/SIMTech_ws/src/microEpsilon_scanControl/launch$ pin
g 169.254.87.67
PING 169.254.87.67 (169.254.87.67) 56(84) bytes of data.
64 bytes from 169.254.87.67: icmp_seq=1 ttl=64 time=0.416 ms
64 bytes from 169.254.87.67: icmp_seq=2 ttl=64 time=0.246 ms
64 bytes from 169.254.87.67: icmp_seq=3 ttl=64 time=0.771 ms
64 bytes from 169.254.87.67: icmp_seq=4 ttl=64 time=0.758 ms
64 bytes from 169.254.87.67: icmp_seq=5 ttl=64 time=0.822 ms
64 bytes from 169.254.87.67: icmp_seq=6 ttl=64 time=0.782 ms
64 bytes from 169.254.87.67: icmp_seq=7 ttl=64 time=0.840 ms
64 bytes from 169.254.87.67: icmp_seq=8 ttl=64 time=0.227 ms
^C
--- 169.254.87.67 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7142ms
rtt min/avg/max/mdev = 0.227/0.607/0.840/0.249 ms
```

Figure 1: Successfully Ping the IP address of Micro Epsilon on linux termianl

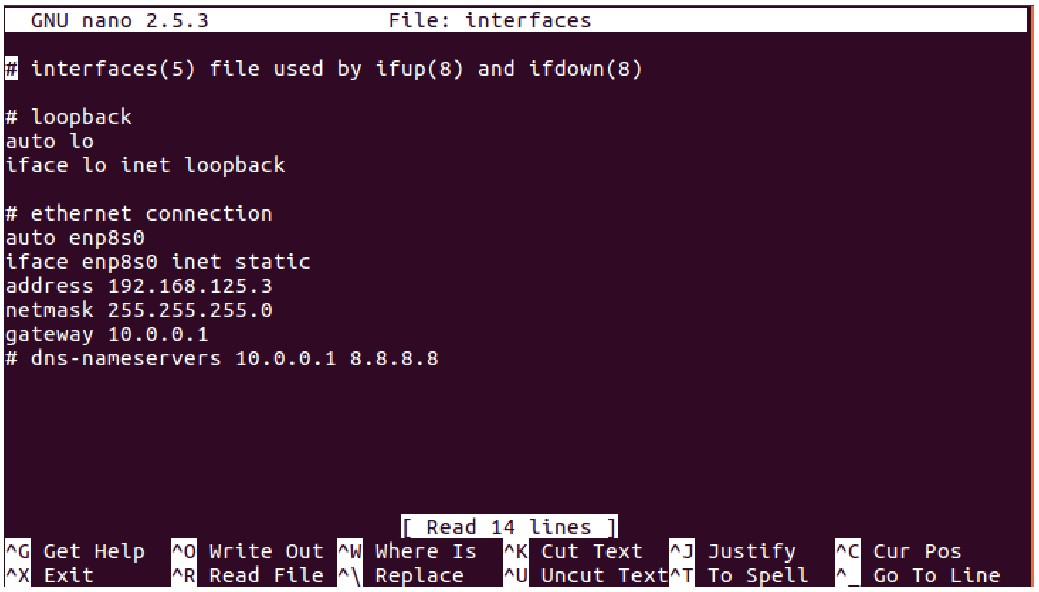
1.2 PC-robot communication

1.2.1 Configure Ethernet settings on Ubuntu for PC-robot communication

- The robot default ip address is: **192.168.125.1**
- If the PC is not successfully connected to the robot after plugging in the ethernet wire: execute following to open the network configuration file:

```
$ cd /etc/network
$ sudo nano interfaces
```

- configure the Ethernet as below, so that the PC can successfully detect the robot controller:



```
  GNU nano 2.5.3                File: interfaces

# interfaces(5) file used by ifup(8) and ifdown(8)

# loopback
auto lo
iface lo inet loopback

# ethernet connection
auto enp8s0
iface enp8s0 inet static
address 192.168.125.3
netmask 255.255.255.0
gateway 10.0.0.1
# dns-nameservers 10.0.0.1 8.8.8.8
```

[Read 14 lines]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line

Figure 2: Ethernet setting for PC-robot communications

Another way to do so is in the first section (the same way as we did for micro epsilon sensor)

- Save the settings and connect the computer to the robot controller irc5
- You shall see the notification **'Your ethernet has been connected'** on the upper right corner on the screen once you have plug in the wire between the computer and robot.
- reboot the network:

```
$ sudo systemctl restart networking.service
```

- ping the robot ip address to see if the connection is successful:

```
$ ping 192.168.125.1
```

1.3 Operating Procedure: Configure programs modules and tasks on robot controller

- Create Tasks (3 tasks): ABB - Control Panel - Configuration - Topics - Controller - Task
  - ROS_StateServer: SEMISTATIC
  - ROS_MotionServe: SEMISTATIC
  - T_ROB1: NORMAL (already exist)
  - ROS_LaserServer: SEMISTATIC

tasks	type	Trust Level	Entry	Motion Task
T_ROB1	NORMAL	N.A.	main	YES
ROS_StateServer	SEMISTATIC	No Safety	main	NO
ROS_MotionServer	SEMISTATIC	SysStop	main	NO
ROS_LaserServer	SEMISTATIC	SysStop	main	NO

- Task functionality:
  - ROS_StateServer – Broadcast joint position and state data
  - ROS_MotionServer – Receive robot motion commands from ROS program
  - ROS_LaserServer — Receive Real-Time controlled value of laser power(current/voltage) and send(setAO) analog signal to the controller
  - T_ROB1 – Issues motion commands to the robot, including MOVEL, MOVEJ, laserON, laserOFF...
- Load program to the task:
  Important things to notice
  - When load the program to the task, you need to set all the task to NORMAL mode and restart the system. After program has loaded, change the socket-server tasks(state, motion, laser) back to SEMISTATIC and restart system again.
  - We have **four different tasks** and **four different programs**
  - Each program corresponds to one task, we need to upload the corresponding program for each task
  - The three programs are stored in three different folders(separate), in order to keep the program clear.
  - Each program contains different modules and have different functionalities. They have a common system file: COMMON.sys, which needs to be loaded to each program.
  - The program should not contain the system file. The system file can only be loaded separately to each tasks.

tasks	Program	Folder	modules	TASK state
T_ROB1	motion_handle.pgf	severmotion	SERVER_motion.mod	NORMAL
ROS_MotionServer	PC_command.pgf	pccommand	SERVER_command.mod	SEMISTATIC
ROS_StateServer	State_feedback.pgf	statefeedback	STATE_raw.mod	SEMISTATIC
ROS_LaserServer	laser_control.pgf	lasercontrol	LASER_control.mod	SEMISTATIC

1.4 Important Note: Change of program in Task

Previously, in the ABB driver part, we also configure the three tasks exactly same as above. In the future, we may also use the same task but load different program to the task. In this way, the program will not conflict with each other. A few things to notices:

- During other operations, we may only use T_ROB1 task and for that situation, we **must not** set the ROS_motionServer and ROS_stateServer to NORMAL(forground). It should only be SEMESTATIC, otherwise the T_ROB1 will not operate.
- For SEMISTATIC task, it runs automatically as soon as the robot controller is rebooted. That means the two server socket(one for receiving joint state one for receiving PC command) are created and keep listening as soon as the controller is turned on. However, The T_ROB1 task, which is set to NORMAL will handles the motion command only if the PC command has been send from the PC client and successfully received by the ROS_motionServer task. **You need to press the 'play' button and push the enable switch in order to run the T_ROB1 task**.
- For debug purpose, you may set the ROS_motionServer and ROS_stateServer to NORMAl mode on the control panel to check if there's any error in the RAPID code. However, remeber to change back to SEMISTATIC and reboot the controller.

1.5 Functionality of each RAPID modules

tasks	Functionality	Corresponding ROS program
SERVER_motion.mod	responsible for issuing commands to the robot executing each command received by motionServer socket translate the command into RAPID code MoveL, Movej, laser on/off	N.A
STATE_raw.mod	a socket server broadcasting the joint state of robot listening to the request sending from PC client socket send joint state data back to the PC client **logger port: 11002**	nd_robot_state.py(client socket) logger_robot.py abb.py
SERVER_command.mod	a socket server listening command send from PC pass the JSON command to the T_ROB1 motion task JSON command includes move linear, move joint, laser on/off, set tool, set workobject etc. **server port: 11000**	nd_robot_command.py(client socket) server_robot.py abb.py qt_path

LASER_control.mod	a socket server listening to the laser power value
	The PC keep updating the controlled power value
	The socket server convert the received value
	Set the laserpower by setAO.
	laser port: 10001
		nd_laser_socket.py(client socket)
		nd_control.py
		control.py(PI controller)
		qt_contro.py(ui)
		/camera_measures/camera_measures.launch
		(image processing)

2 Control Experiments

2.1 I/O Data collection

In Order to Know the control parameters and predefined values for minor axis(reference value), we start the launch file which contains the camera measurements nodes.

- launch file:

```
$ roslaunch simtech_robot_laser_control robot_laser_control.launch
```

By using the ROS bag tools, we can easily record data, display data, and republish data. Here is the procedure of using rosbag for data acquisition:

- record the specific topic into a bag file

```
$ rosbag record /usb_cam/geometry
```

- or in a better way :
  reference: `http://wiki.ros.org/rosbag/Tutorials/Recording%20and%20playing%20back%20data`
  The -O argument tells rosbag record to log to a file named subset.bag

```
$ rosbag record -O subset /usb_cam/geometry /control/power
```

- record all the topics

```
$ rosbag record -a
```

- Output the bag information

```
$ rosbag info -y <bag_file>
```

- to view the content of the bag file, to plot or to display data;

```
$ rqt_bag
```

- to save the data from bag file to txt file

```
$ rostopic echo -b file.bag -p /topic > data.txt
```

2.2 Step Experiments for system identification

Matlab analysis to get the system transfer function

2.3 Velocity-Power control

- launch the program:

```
$ roslaunch simtech_robot_laser_control robot_velocity_power_control.launch
```

- record the topics

```
$ rosbag record -O Name /velocity /acceleration /twist /twist_acceleration /position /control/power
```

- after the experiment: extract the data from bag file:

```
$ rostopic echo -b Name.bag -p /velocity > velocity.txt
```

  same for the other topics, this step will move the data inside the bag file to a txt file

2.4 Acceleration-Power control

- launch the program:

```
$ roslaunch simtech_robot_laser_control robot_acceleration_power_control.launch
```

- record the topics

```
$ rosbag record -O Name /velocity /acceleration /twist /twist_acceleration /position /control/power
```

- after the experiment: extract the data from bag file:

```
$ rostopic echo -b Name.bag -p /velocity > velocity.txt
```

  same for the other topics, this step will move the data inside the bag file to a txt file

2.5 Meltpool-Power PID control

- launch the program:

```
$ roslaunch simtech_robot_laser_control robot_laser_pid_control.launch
```

- record the topics

```
$ rosbag record -O Name /usb_cam/geometry /control/power
```

- after the experiment: extract the data from bag file:

```
$ rostopic echo -b Name.bag -p /velocity > velocity.txt
```

  same for the other topics, this step will move the data inside the bag file to a txt file

3 Surface scanning and defects identification experiments

3.1 Operating Procedure: scanning the surface(Traning Data collection)

- Launch the Program:

```
$ cd ~/SIMTech_ws/src/scanning_application/scanning_robviz/launch
$ roslaunch scanning_robviz robviz_microepsilon.launch
```

- click the record data button
- the point cloud data is recorded and stored in folder **data** under scanning_robviz

3.2 Parameter used during multi-plannar segmentation

– on linux terminal (PCL_filtering tool), executes:

```
$ ./PCLFiltering_tool -load surface2.pcd -save surface2_segmented.pcd
 -multiPlannarSeg -DistanceThre 0.001 -Stddev 1.4
```

where distance threshold specifies how far a point is allow to be away from the plane model. standard deviation factor is for statistical filtering.

3.3 Producing feature data for machine learning training

- -previously, from file **SIMTech_ws/src/RT_monitoring_application/Point_cloud_visualziation/ PCLFIltering_Tool/PCLFiltering_tool.cpp**
  now changed to:

```
$ cd ~/SIMTech_ws/src/scanning_application/Point_cloud_visualziation/PCLFIltering_Tool
```

- Features to be extracted: Coefficient of the plane, Point to Plane distance and Point normal vector calculation
- to execute the multi-plannar segmentation program need to choose correct parameters, in the console:

```
$ cd build
$ ./PCLFiltering_tool -multiPlannarSeg -load surface1.pcd -save plane1_segmented.pcd
-saveCoefficientPlaneName coefficient_plane1.txt
-savePointToPlaneDistance PointToPlaneDistance_plane1.txt
-saveNormalEstimation NormalEstimation_plane1.txt
-DistanceThre 0.001 -leafsize 0.001 -Stddev 1.2
```

- Distance threshold parameter (specifies the maximum distance allowed between a point to the plane model, if it is out of range, it will be filtered and not be regarded in this plane) needs to be changed accordingly to satisfy the segmentation conditions for different planes. Some may need larger distance threshold if they are defects plane.
- The method for calculating point to Plane distance and Point normal vector estimation is explained below.

3.4 Experiment after traning has complete: defects prediction

- To use only microepsilon, without connecting robot (for testing purpose):

```
$ cd ~/SIMTech_ws/src/scanning_application/scanning_robviz/launch
$ roslaunch scanning_robviz robviz_microepsilon_simulation.launch
```

- To use the micor-epsilon sensor together with real robot :

```
$ cd ~/SIMTech_ws/src/scanning_application/scanning_robviz/launch
$ roslaunch scanning_robviz robviz_microepsilon.launch
```

- Change of parameters in nd_subprocess_handler

```
// -------------------subprocess start(segmentation process)--------
std::string executable = path + "/PCL_segmentation/build/PCL_segmentation";
std::string option = " -multiPlannarSeg";
//other options: -NormalSegmentation, -largestPlane, -ShapeSeg, -sfilter, curveSeg
std::string loadfile = " -load " + path + "/pcl/" + this->pcd_filename;
std::string savefile = " -save " + path + "/pcl/" + this->pcd_segmented_filename;
std::string savePointDistance = " -savePointToPlaneDistance " + path + "/distance/" + this
std::string parameters = " -DistanceThre 0.001 -Stddev 1.5";
// std::string parameters = " ";
// Initialize String Array
std::string command_line = executable + option + loadfile + savefile +
savePointDistance + parameters;

auto p = sp::call({command_line});
//--------------------subprocess end------------------------
```

- To test the stand alone functionality of the **PCL_segmentation**:

```
$ cd ~/SIMTech_ws/src/scanning_application/scanning_robviz/PCL_segmentation/build
$ ./PCL_segmentation -multiPlannarSeg -load 1.pcd -save 1_s.pcd
```

```
-savePointToPlaneDistance distanc.txt -DistanceThre 0.001
-Stddev 1.5
```